Deep learning workshop

Summary and homework

In Li's philosophy of 'not reinventing the wheel', the first assignment is watching this video:
https://www.youtube.com/watch?time_continue=46&v=iaSUYvmCekI&feature=emb_logo
Special thanks to Hanka for suggesting this video!

In the dropbox folder you can find the python script 'classify_mnist.py'. This is an example of how to create a convolutional neural network and train it on the handwritten digits (MNIST) dataset. It only uses the torch library. Feel free to play around with the code. You can, for example, add layers, alter the number of neurons, or change the activation functions. It should be small enough that most pc's/laptops can handle it, but I do not expect you to have run it, just studied the code a bit.
Tip:
Although the handwritten digit dataset (MNIST) is a small fun set to experiment with, be careful of the conclusions you draw. If a new method works very well on MNIST, it does not necessarily works equally well on other datasets. If you come across papers where they test their method only on MNIST, it is very likely it doesn't generalize to other datasets.

Most convolutional neural networks have convolution layers followed by a max pool and an activation function:

```
self.pool(F.relu(self.conv1(x)))
```

Question 1: Does applying the activation function **after** the maxpool (instead of **before)** give different results?

After the convolution and pooling layers, often one or more fully connected layers are added to the network. To make the output of the convolutional layer compatible with the linear layers, the output shape is transformed with the **view** function or the **nn.Flatten** class.

In pytorch there are two ways to define the same convolutional neural architecture:

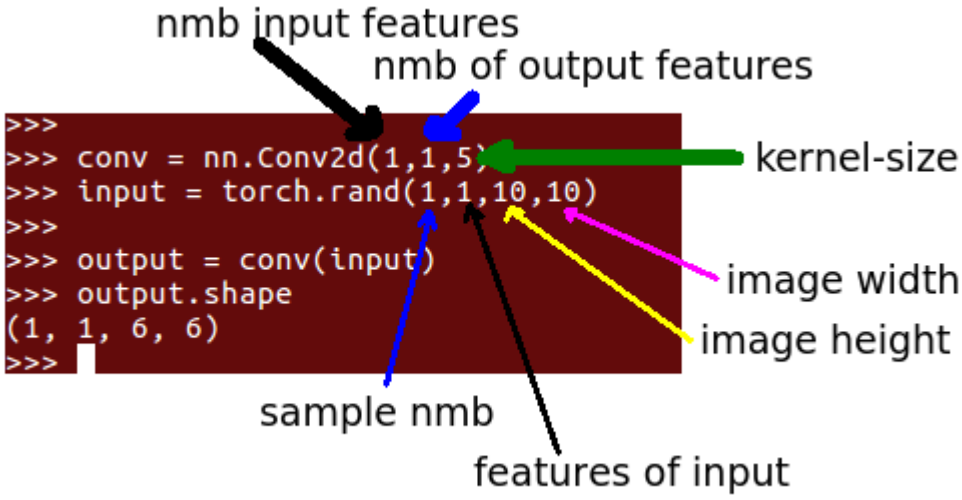
```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 8, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(8, 16, 5)
        self.fc1 = nn.Linear(16 * 4 * 4, 256)
        self.fc2 = nn.Linear(256, 64)
        self.fc3 = nn.Linear(64, 10)
    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 4 * 4)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.softmax(x, dim=1)
```

```
Net = nn.Sequential(nn.Conv2d(1, 8, 5),
                    nn.ReLU(),
                    nn.MaxPool2d(2,2),
                    nn.ReLU(),
                    nn.MaxPool2d(2,2),
                    nn.Flatten(),
                    nn.Linear(16*4*4, 256),
                    nn.ReLU(),
                    nn.Linear(256, 64),
                    nn.ReLU(),
                    nn.Linear(64, 10),
                    nn.Softmax(dim=1))
```

Fun fact: The creator of convolutional neural networks does not like the abbreviation CNNs but prefers ConvNets

Question 2: The neural network of the previous question does not contain batch normalization. Batch normalization comes in three flavors: nn.**BatchNorm1d** nn.**BatchNorm2d** and nn.**BatchNorm3d.**
Two of these can be used in our network, which ones and where are they allowed in the network?

Question 3a:
The ConvXd is the main function of convolutional neural networks. Below you can see an example of this function:




```
>>>
>>> conv = nn.Conv2d(1,1,5)
>>> input = torch.rand(1,1,10,10)
>>>
>>> output = conv(input)
>>> output.shape
(1, 1, 6, 6)
>>>
```

Notice that the input tensor is 10 x 10 pixels, while the output is 6 x 6.
Why is the output smaller than the input?

Question 3b:
If the kernel-size is changed from 5 to 1, will the input and output tensors still differ in shape?

Question 3c:
Sometimes you want the output tensor of a convolutional layer to have the same shape as the input tensor. The common solution for this is using padding as can be seen below:

```
>>> conv = nn.Conv2d(1,1,5, padding=2)
>>> input = torch.rand(1,1,10,10)
>>>
>>> output = conv(input)
>>> output.shape
(1, 1, 10, 10)
>>>
```

Search online what padding is (in the context of neural networks) and why this solves this problem.

Question 4:
The hard programming assignment :)
Here we are going to build a convolutional neural network capable of translating the most used language in the world. Namely, DNA sequence to protein sequence. Lets pretend we forgot the codon table and want to use a neural network to learn how to translate DNA to protein. The input will be a DNA sequence of arbitrary length, and the output the corresponding protein sequence.
Data: The complete dataset will be three DNA sequences with the corresponding protein sequences. These can be found on the last page or in the ‘w5.fasta’ file in the dropbox folder.

Preprocessing: As discussed before, neural networks can not accept character-symbols as input, only numbers. Therefore the nucleotides and aminoacids have to represented as numbers. A popular choice is representing them as one-hot encoded vectors. Take the nucleotides of DNA for example:

A → 1, 0, 0, 0 **T → 0, 1, 0, 0** **G → 0, 0, 1, 0** **C → 0, 0, 0, 1**

The same can be done with the amino-acids.

4a: Start with creating functions that can transform DNA and protein sequences into one-hot vectors. Also create a function that can transform probability distributions (the output of your network) back into a protein sequence to check the results of your neural network.
Tip: use the **argmax** function to find the highest values in your distribution.
The shape of an one-hot sequence should be: (1, number_of_input_features, sequence_length)

4b: Use the following code to create your neural network:

```
net = nn.Sequential(nn.Conv1d(?, ?, ?, stride=3),
                    nn.BatchNorm1d(?),
                    activation function of choice
                    nn.Conv1d(?, ?, 1),
                    nn.BatchNorm1d(?),
                    activation function of choice
                    nn.Conv1d(?,?, 1),
                    nn.Softmax(dim=1))
```

Notice that sequences are 1 dimensional, and therefore we use Conv1d and not Conv2d.

4c: Find out what **stride** is in convolutional neural networks.

4d: Create code to train the network. Do this by translating the following pseudo-code into python:

```
Do 1000 epochs:
      for all nucleotide and protein sequence pairs:
            clear gradients with optimizer
            prediction = neural_network(nucleotide_sequence)
            calculate the loss with binary cross entropy
            loss.backward()
            apply gradients with optimizer
      print loss over the three sequences
```

4e: When you saw the thee DNA sequences, you of course immediately noticed that only 2 of the 4 proline codons where there. What does your network predict if you give all proline codons, and how do the probability distributions look like? (proline codons: cct, ccc, cca, ccg)

4e: Which of these inputs can **not** be used as input for your convolutional neural network and why?

Tensor **A** of shape (1, 4, 3)
Tensor **B** of shape (1, 5, 9)
Tensor **C** of shape (1, 4, 6)
Tensor **D** of shape (1, 1, 7)
Tensor **E** of shape (1, 4, 7)
Tensor **F** of shape (1, 4, 2)

Question 5: watch: https://www.youtube.com/watch?v=8yFQc6elePA for an example where convolutional networks can fail.

> Sequence 1 DNA
ATGATTACTTGGCTTACATTGGATCTGGCGTTTAACGGCATGGTAACGTGGATGATCAAATTTCATAGTTGTGA
AGCTCCCAGGTCCGATGTCTTGCCCATGACATCAACTGCCGGTCGAATATCGCAGCAGATGTTCGACTTCACCG
TAACCAACTGGGAAGTTTTCCGGGTGACTCAAAGGGGAGCTGTGGGACACACAATAAAGGCAATACCTCCTATA
CCCAGAGGAATTTTATTTTGGCTAGCGGTGGTAATGTGTCACTCCTTGTGCATTTCTAGGCGATTCATCAAACA
GACCCATTAG

> Sequence 1 PROTEIN
MITWLTLDLAFNGMVTWMIKFHSCEAPRSDVLPMTSTAGRISQQMFDFTVTNWEVFRVTQRGAVGHTIKAIPPI
PRGILFWLAVVMCHSLCISRRFIKQTH*

> Sequence 2 DNA
ATGAGATTTTTCCCCCGCAAAGTTGAGAACTTCTCTCCCGTAAGAAGCGGCATGCAAATACGCGATCACCATAA
TCTAAAGGAGGTCAATTCGTGGGAGCATGATGAGTGTGTGCTCGCTTGTGCTGAGCTGATGTTTTTTCGTGCCA
CTGGAGTGTGGATTTGTCCCGATATGCTATATTGCACTGAGTATGTCATACGCGGGGTGATGTCTTGCTGGGAG
ATGGACGATATGCAGATTCACATAGCGAAATGGAATAGAACTTGTGGATGGCACCCTGTAGACAAGCCTCTCTA
TATCGAGGACGGTGACACGCCCTCGAGTCGATTTGGCGGCACATGGATCTTCATCGCCTCGGCCGTTATCCACG
AAAGCTGGCATACGATGTGGAAGGAAGTCTTCAAAATGCACATGTCCTCGTTTGCACATTGCGTAGTGATACAT
TGGTATATGTCGTGTATCGCAGAGGGGCAAACCTGTATGAACGGGCAGTCCTATTTCGCAATAATTCCCGTACC
TCAGAACACCTATAATGAATGCCTTTGTAGAAAGCAGTCCTGGAATACACCTGCGAATTACGTAGCGCCTGGGA
TCCATCACTGGTTTATGTGGCAGATAGGCATGCCCCAATTTCACGAACATGGCGACACCATGTTCTTAATGTTT
GGAGTTGAAATGCCCGCTATATACGTTCAAGAAAAATGTTGGATAGTTACAGGAAAACCTAAATACCCTGAAAC
AATCATTTGGATTGTCTGCCATGTTATGACCGGGTGTCCCGACGTCCAACTAATGATAGCAGATCATCCTTATA
ACTGTCATCAAGTCGACGATGAGCAAGTTCCCATCCTATCATGTGAATTCCACTGTTGGAACGACGTCATCTTT
TATTTGACGATTGCTGTGAATATGCGTACAGACAGGTTCCAAGCAACAATGCCCTTCTGGAACTTCCAAAGGAA
GCAGAATTACGACAATGACATGCATAAGAATTCCTGCGAAAGTCCTCTCTCCCACGAATATCAGCAGATGTACC
GTACCGAGGCAATTGCATAA

> Sequence 2 PROTEIN
MRFFPRKVENFSPVRSGMQIRDHHNLKEVNSWEHDECVLACAELMFFRATGVWICPDMLYCTEYVIRGVMSCWE
MDDMQIHIAKWNRTCGWHPVDKPLYIEDGDTPSSRFGGTWIFIASAVIHESWHTMWKEVFKMHMSSFAHCVVIH
WYMSCIAEGQTCMNGQSYFAIIPVPQNTYNECLCRKQSWNTPANYVAPGIHHWFMWQIGMPQFHEHGDTMFLMF
GVEMPAIYVQEKCWIVTGKPKYPETIIWIVCHVMTGCPDVQLMIADHPYNCHQVDDEQVPILSCEFHCWNDVIF
YLTIAVNMRTDRFQATMPFWNFQRKQNYDNDMHKNSCESPLSHEYQQMYRTEAIA*

> Sequence 3 DNA
ATGTGCATCCAAGCCTTGCTCTTTCACCAGAAAAACGAAGCCCTGATACACTTTCACTTCGGAAGTGCGCACAC
GTCGATTGAGGGAATGCCCAAATTACCCGGGAAGTCGGTCGACGCGACGATTGCCTTCATGCAATACCAGGTAT
TTGATACTCATAACGCGAATACGAATGACACTTATGAGTGGATAGATGAAATGAATATGAAACACGACCAAACA
GACATTATGACCGCTGCCATGCTAAATTTCCACGGAGGTCTCAGTTGTTATGTGACAAAATCTATGAACGATCA
GAAGGACTTCCAGATCGAGCCCCAATACTCGATGTCTAGCCCTGAGGACGGTTACGTAGTGTTCCGCCATAATG
TTGGTCTTCATCCTTTTAAGCACAACATTGGCGGAAACCAAATTCAGTGCCCCCCTATACACGCAGACCTTTGG
AAAAAAAACGCCGTATGTAAGTGTTATCTCTATGACCATGAACATTCTATGAACCACCAGTTCATTAACTTTTG
GAGGGTGCATAACCAGCTAGGGCGGGATCCTCAACACCTTAAAAAGCCTTGGCAGGCACGGAAACATACACAGG
AGAGCGACAATGTCCAACCTACCATAGTACCTAATGACAAATTTAATATTACTCAAATCCACGCGTTCTGCTTT
GGGATGTACTCCGTATTGATTAAGTGGTGA

> Sequence 3 PROTEIN
MCIQALLFHQKNEALIHFHFGSAHTSIEGMPKLPGKSVDATIAFMQYQVFDTHNANTNDTYEWIDEMNMKHDQT
DIMTAAMLNFHGGLSCYVTKSMNDQKDFQIEPQYSMSSPEDGYVVFRHNVGLHPFKHNIGGNQIQCPPIHADLW
KKNAVCKCYLYDHEHSMNHQFINFWRVHNQLGRDPQHLKKPWQARKHTQESDNVQPTIVPNDKFNITQIHAFCF
GMYSVLIKW*